A Model-Based Programming Skunk Works

Andrew Bachmann, Charles Neveu,
Charles Pecheur, Mark Shirley,
Will Taylor, Steve Wragg,
Patrick Regan, Louise Helenius

Previously: Brian Williams & Reid Simmons

Summary

Project Type:

Infrastructure and support

Goal:

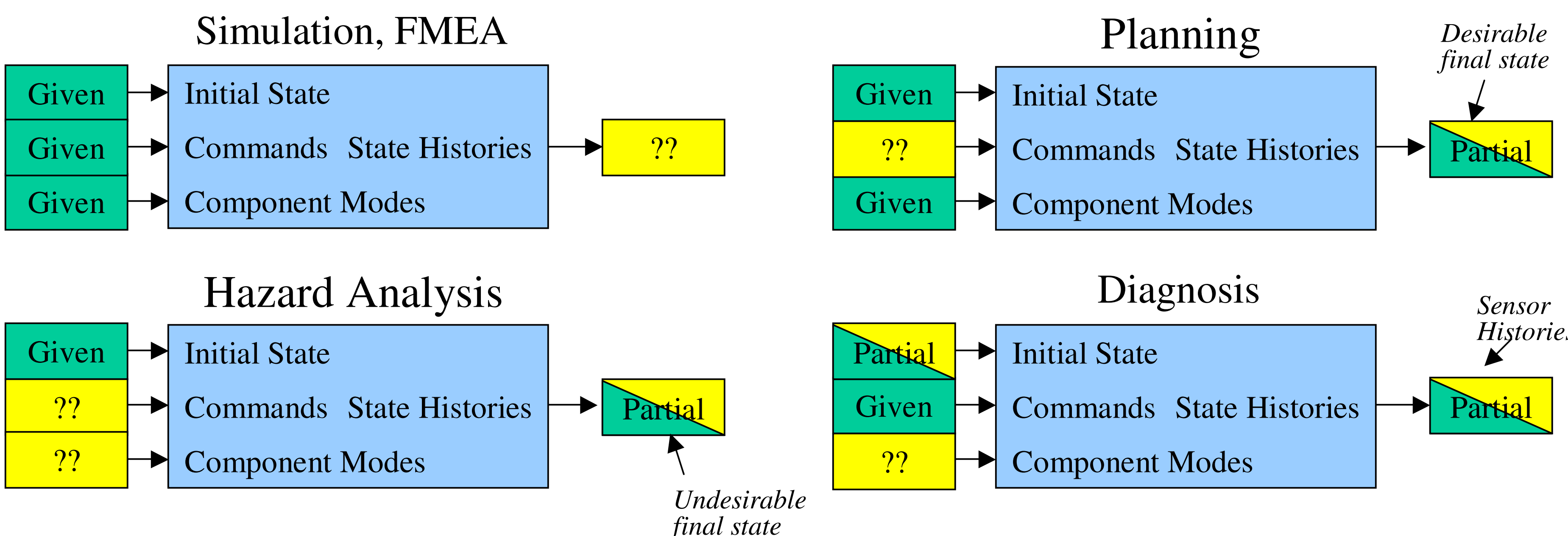
Create development & debugging tools that enable a small team of spacecraft engineers to rapidly create high capability autonomy software

Status:

- Work focused on fault detection, identification & recovery
- Key goals achieved (but similar work needed for rest of agent)
- Project ending this year
- Proposals for two, smaller follow on tasks

Model-Based Programming

- Build a mathematical system model:
  Describe what the system *can* do (the artifact) separately from what you *want* it to do (the control policy)
  - Greatly facilitates model reusability
- Analyze this model mechanically to find ‘goal’ behaviors, depending upon the analysis task
  - Simplifies programming control code by accounting for the combinatorics of component interactions



Component-connection models

Original Project Goals

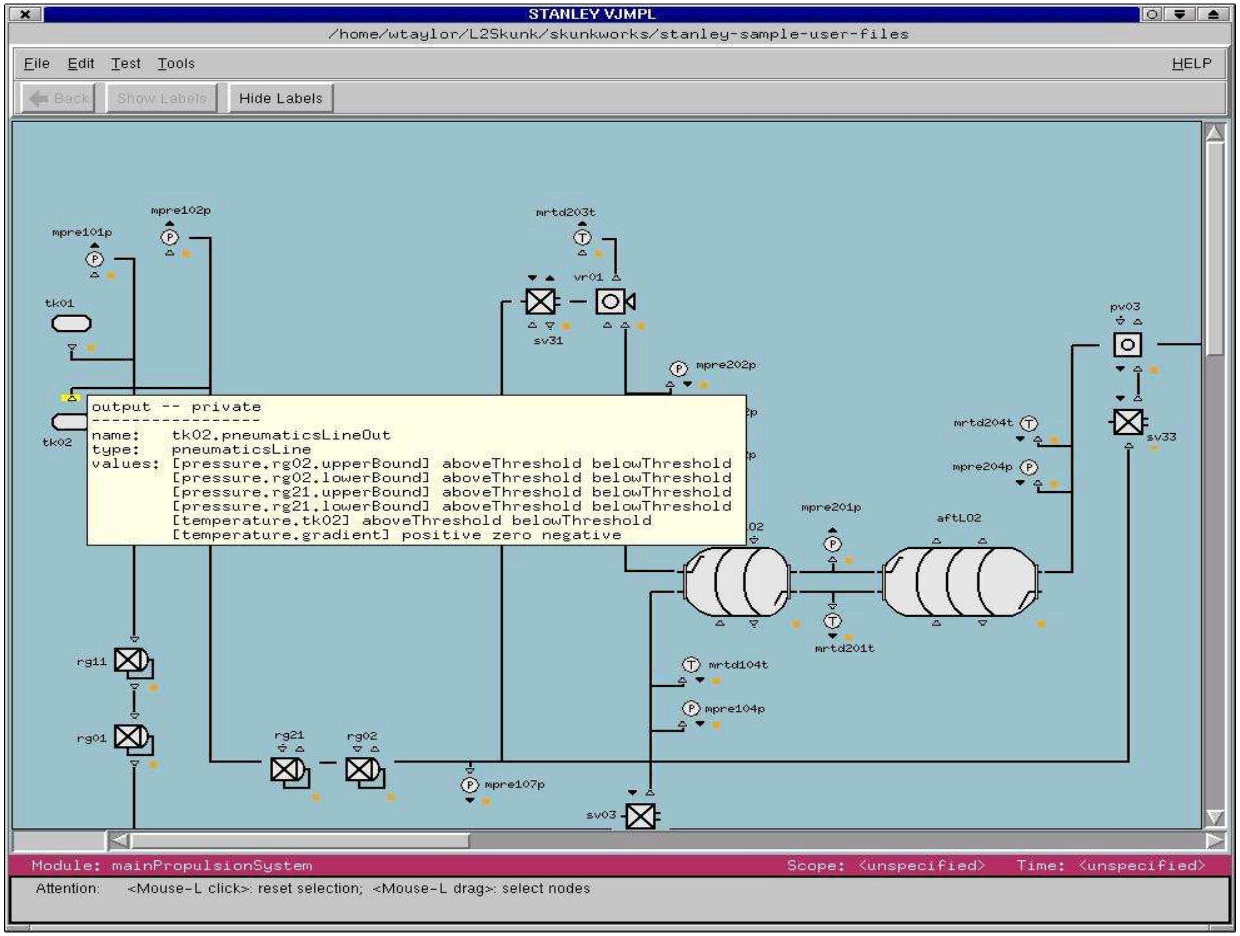
1. A declarative, engineer-friendly model-based programming language
2. A visual model development environment
3. Tools for automatically generating test model procedures
4. Tools and processes for collaborative model development
5. Validation through a pair of autonomy experiments conducted by spacecraft engineers and university graduate students

Goal 1. An Engineer-friendly Model-based Programming Language

- Developed JMPL (Java-MPL)
- Object oriented, has a Java-based syntax
- Compiles model to XMPL format
  - XML-based model interchange language used by Livingston & Northrup/Grumman RLV2 team (spec available)
  - proposed as a model interchange format for L2, Titan (Williams, MIT) and derivatives

Goal 2. A Visual Model Development Environment

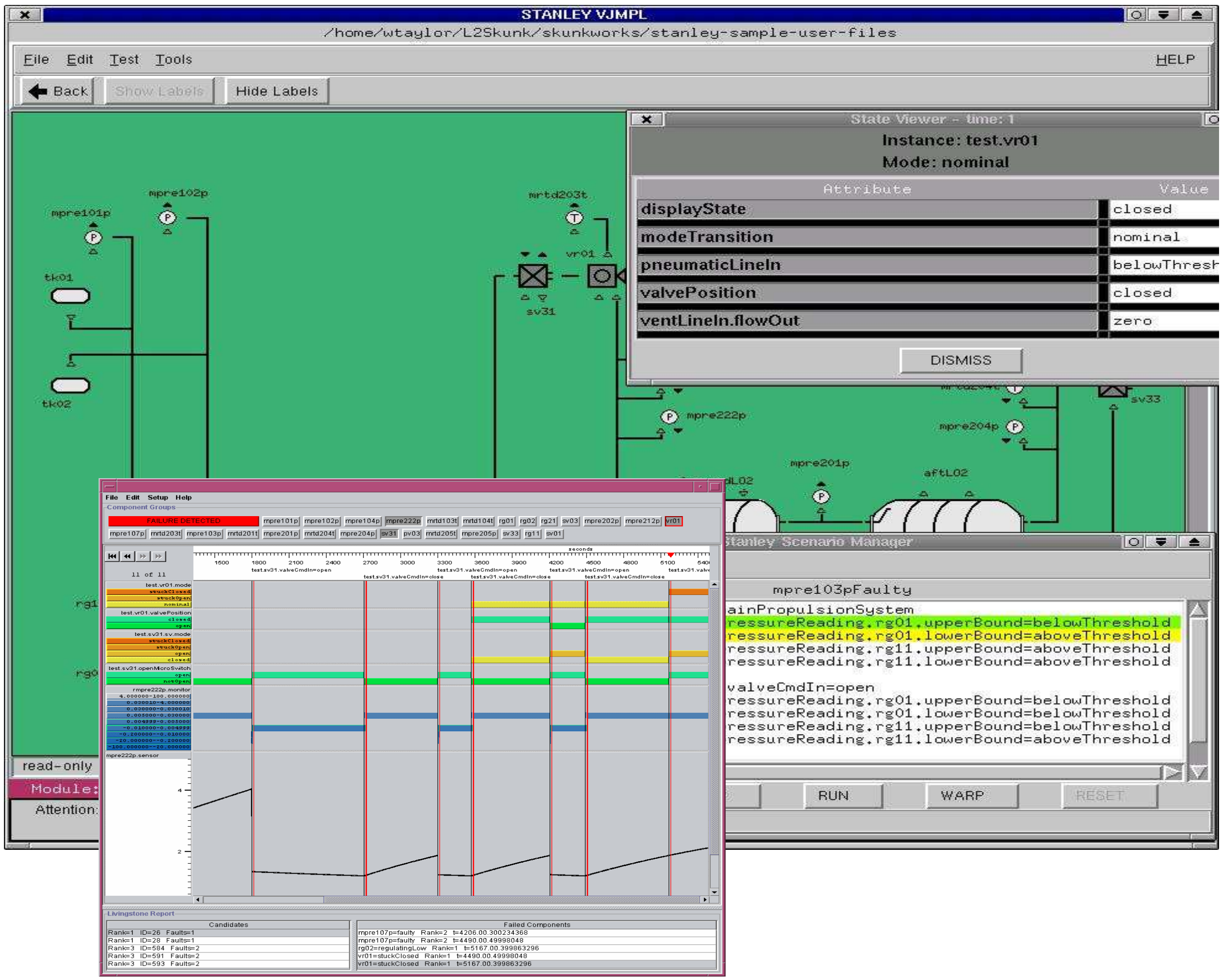
System Modeling



- Stanley (initiated under RAX)
- Completed under Skunkworks
- Visual modeler
- Component Library
- Draw schematic
- Draw state machines describing individual components
- Add constraints as JMPL code fragments

Goal 2. A Visual Model Development Environment

Scenario Debugging



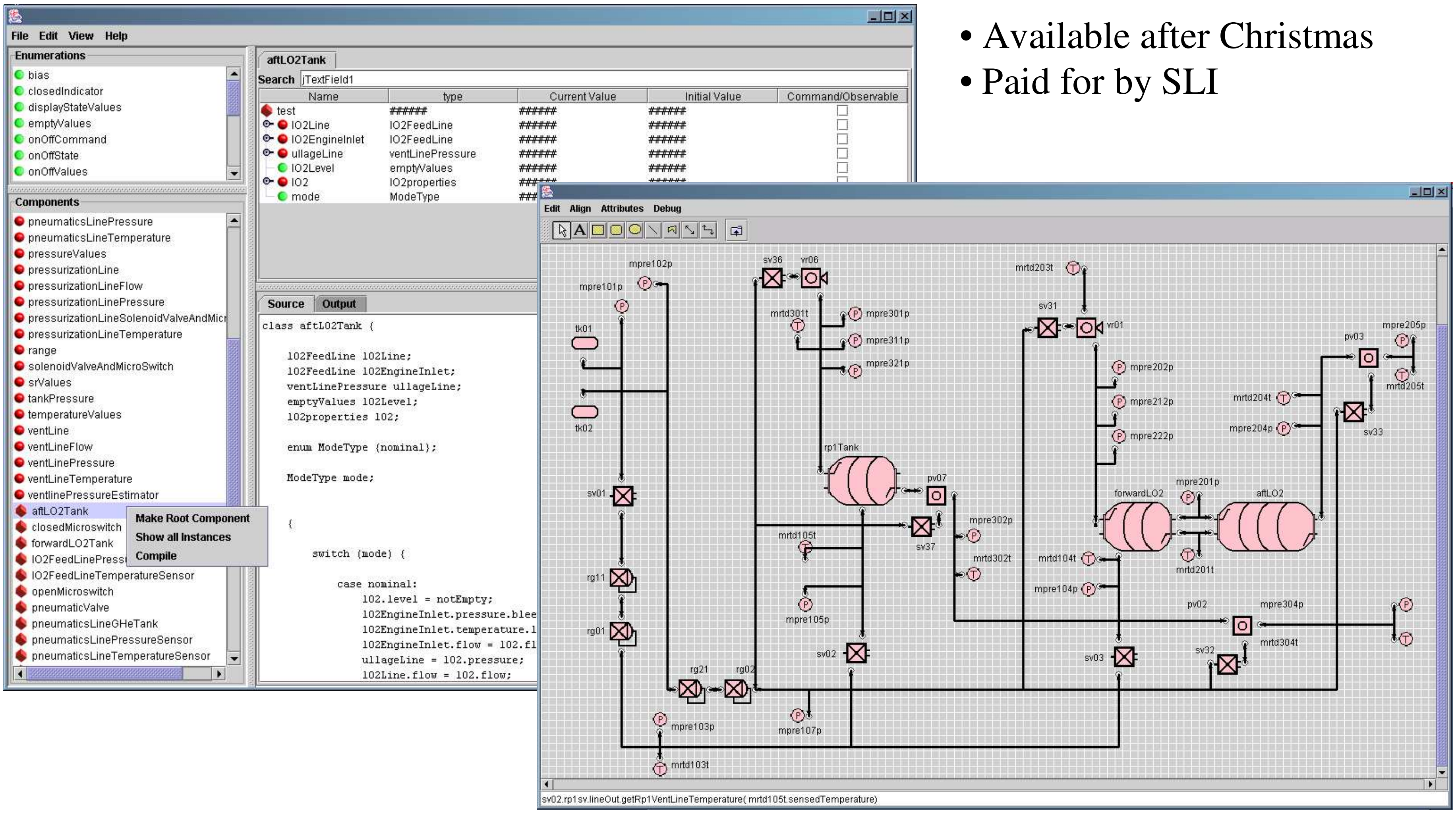
PITEX GPU Parameter History Display

- Invoke compiler with selected model JMPL code to generate XMPL code.
- Interactively with Scenario Mgr, or with editor, create test scenarios.
- Load XMPL model into Livingstone (L2).
- Use Scenario Mgr to send cmds to L2.
- Update Stanley display with L2 state.
- Interact with Candidate Mgr & History Table.

Included in Livingston release

Goal 2. A Visual Model Development Environment

Finally started a reimplementation on a more maintainable foundation



- Available after Christmas
- Paid for by SLI

Goal 3. Tools for automatically generating test procedures for models

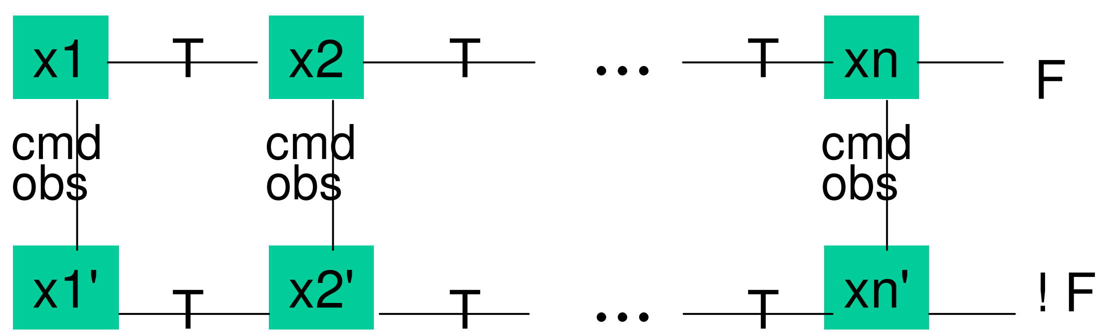
- Shifted from test generation approach to model-checking
- Two approaches
  a. Translation of Livingstone model to a model-checker (SMV)
  b. Explicit search of execution traces using Java Pathfinder (Automated Software Engineering group at ARC)

a. From Livingstone Models to SMV Models

- Developed by Charles Pecheur (Ames) and Reid Simmons (CMU)
- Similar nature => translation is easy
- Properties in temporal logic + pre-defined patterns
- Two generations: MPL (lisp) & JMPL (java)
- Supports model consistency check & limited forms of hazard analysis
- Experiments with ISPP (KSC)
  - Huge state space (10^55) but tractable with SMV
  - Exposed known and unknown modeling errors

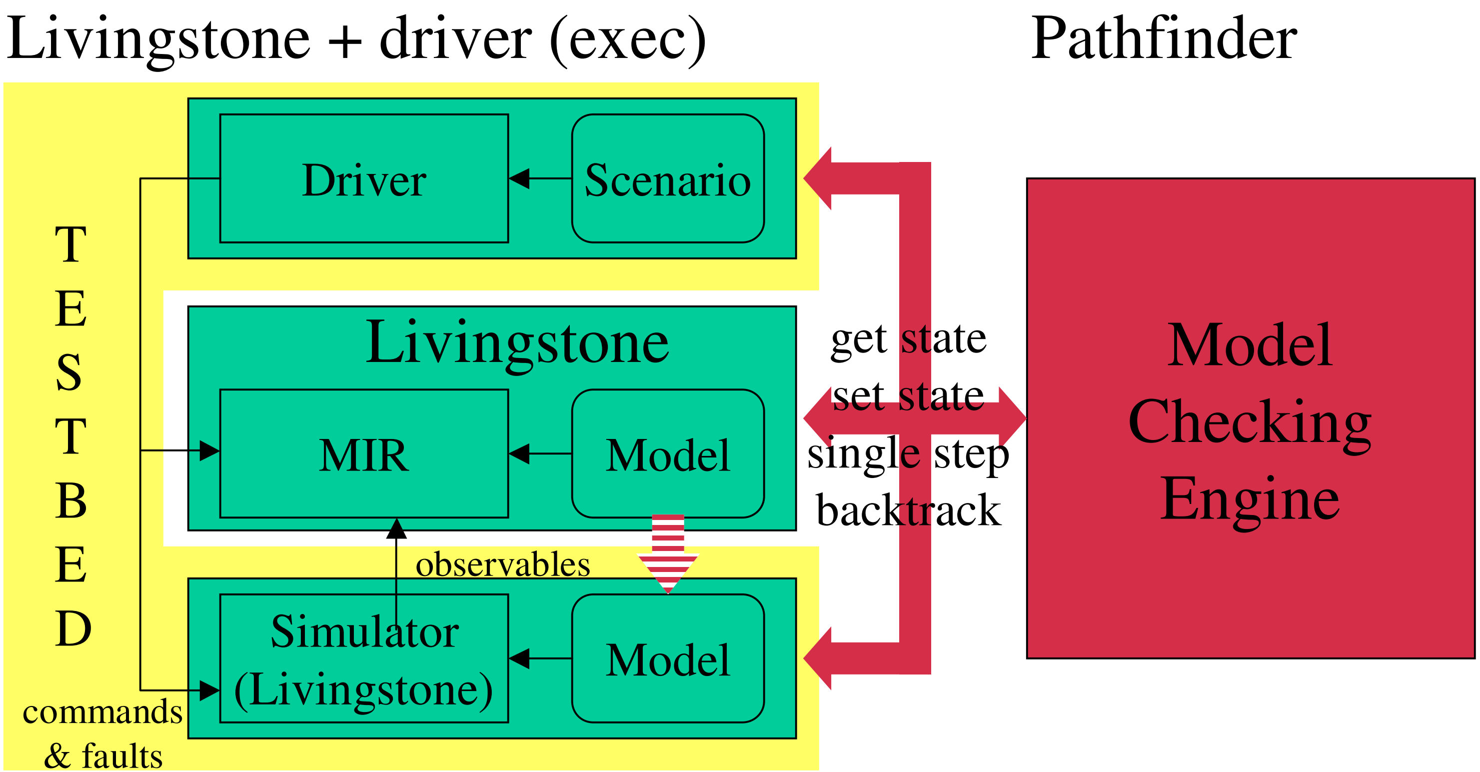
a. Assessing Diagnosability

- Can fault F be diagnosed knowing the last n steps (assuming correct model and "perfect" engine)?
- Look for two sequences (of length n), one ending in F and not the other, that look identical to diagnosis (same commands and observables)
- Approach: use SAT solver to find them



Paper available

b. Livingstone Pathfinder



- Start from conventional testing (the real program).
- Instrument the code to be able to do full model checking (or as close as possible).

Continued under ECS

Goal 4. Tools and processes for collaborative model development

- Nothing special done
- We’re using standard tools like CVS, GNATS …

Goal 5. Validation

Customers:

- X-34 Experimental Reusable Launch Vehicle (NITEX/PITEX experiment)
- X-37 Experimental Reusable Launch Vehicle
- Honeywell and Interface Control Systems RLV2 team
- Northrup/Grumman RLV2 team

All associated with NASA's Space Launch Initiative

Efforts outside of monitoring & diagnosis

- Plan library development tools (last 6 months)
  - Designed and partially implemented new language for Europa (NDDL)
    - Andrew Bachman, Jeremy Frank, Ari Jonsson
  - Implemented 'Potato' visualization of the planning process (moving toward planning process visualization toolkit)
    - Will Taylor

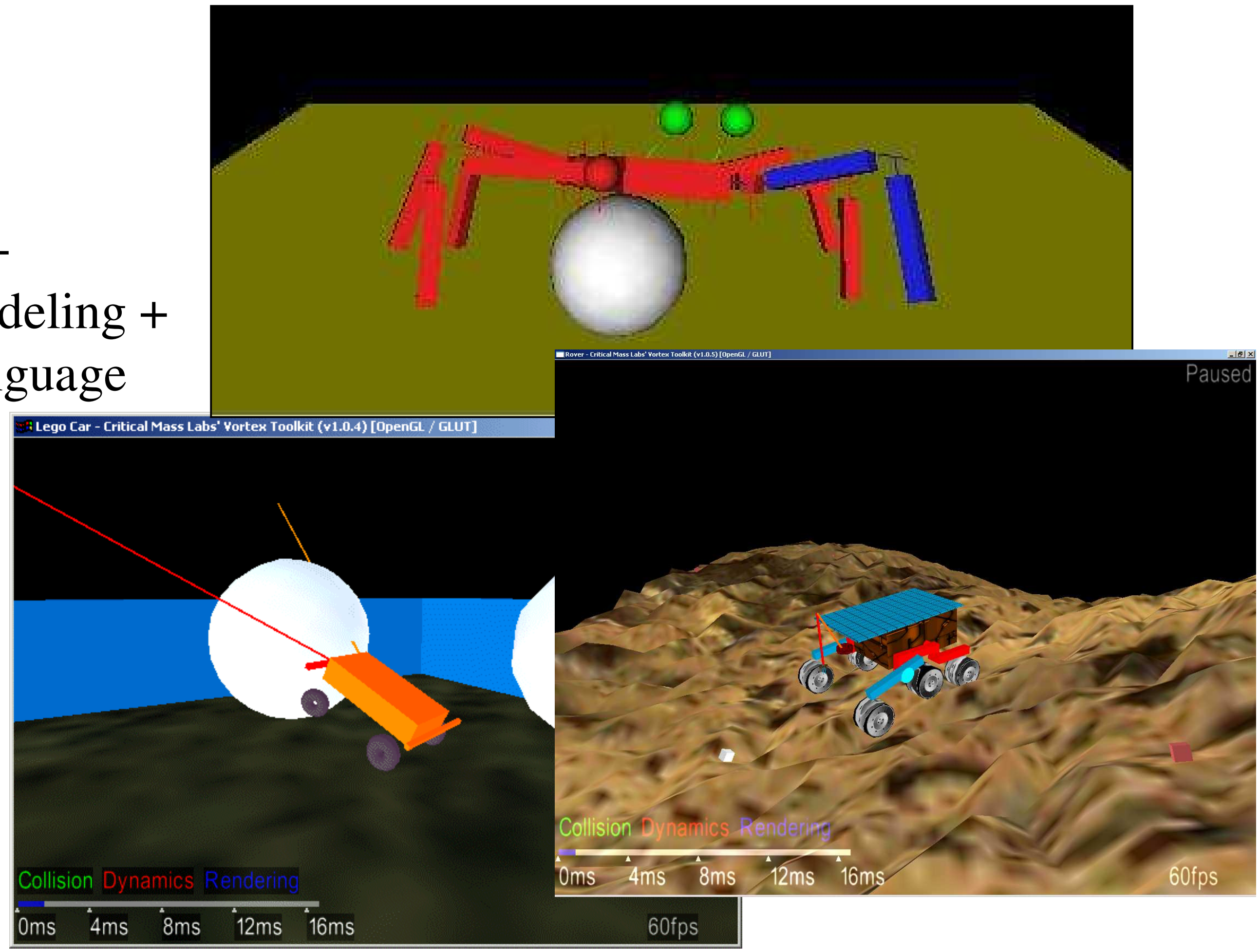
Efforts outside of monitoring & diagnosis

- Rapid prototyping of autonomy testbeds

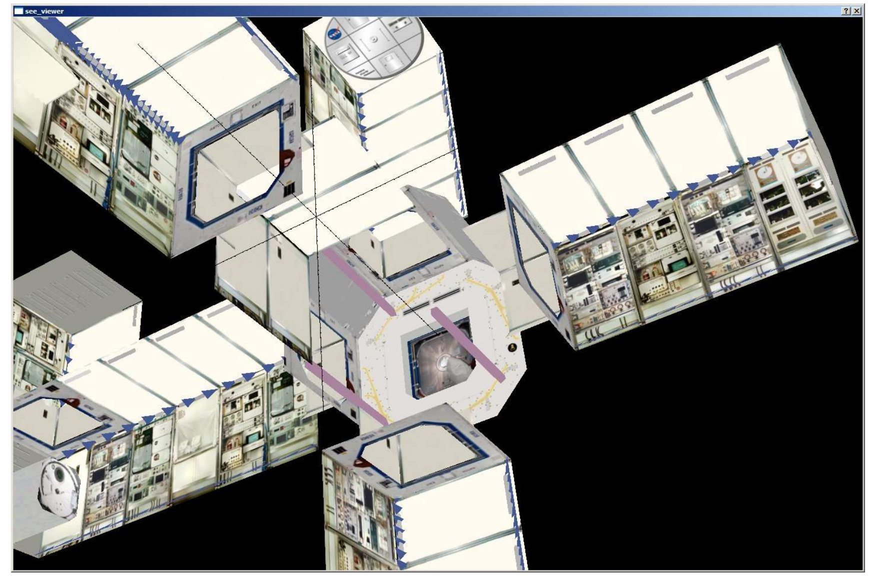
LiveInventor

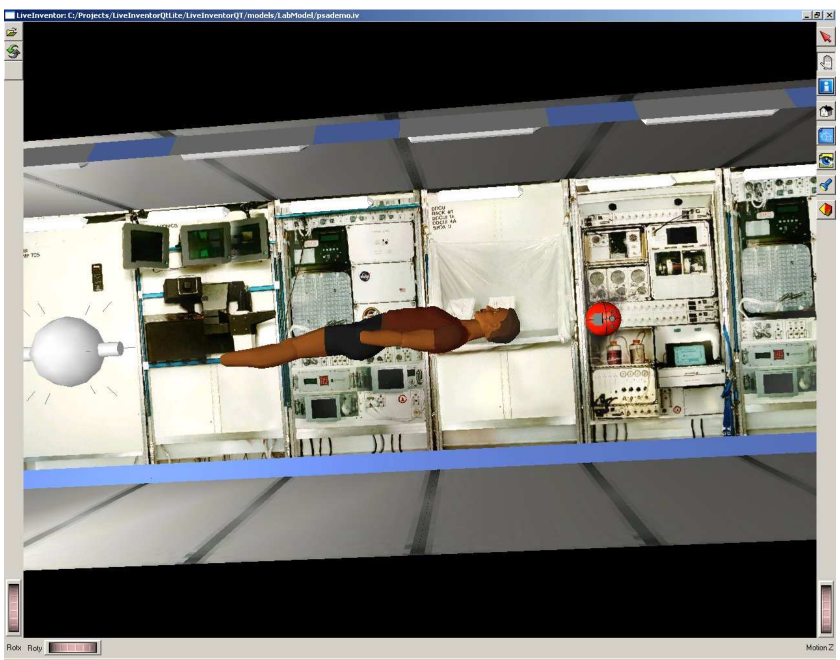
dynamics +
kinematics +
collisions / friction +
integrated world modeling +
hybrid execution language

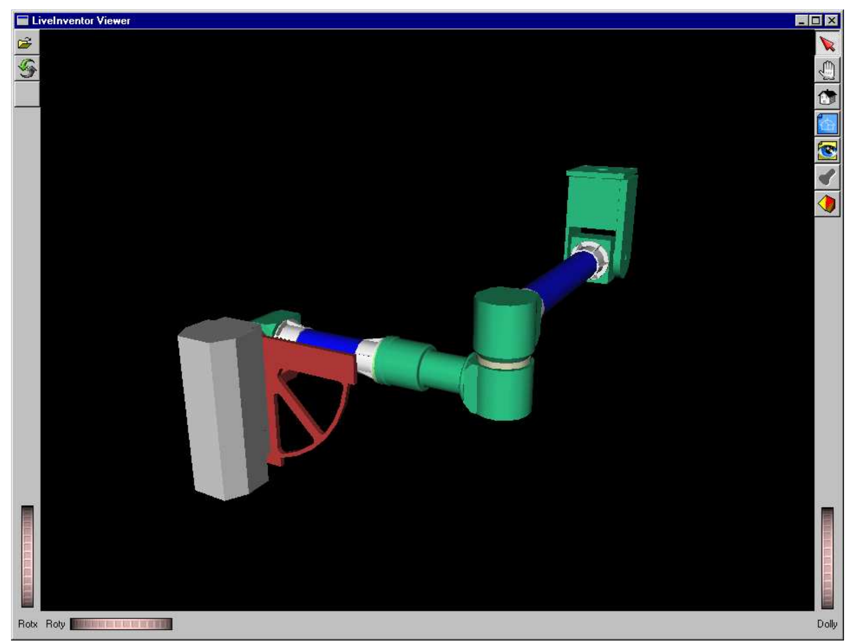
Charles Neveu,
Mark Shirley




LiveInventor: C:/Projects/LiveInventorQtLite/LiveInventorQT/models/LabModel/psademo.iv
Rotx Roty
Motion Z


see_viewer


LiveInventor: C:/Projects/LiveInventorQtLite/LiveInventorQT/models/LabModel/psademo.iv
Rotx Roty
Motion Z


LiveInventor Viewer
Rotx Roty
Dolly

A Model-Based Programming Skunk Works

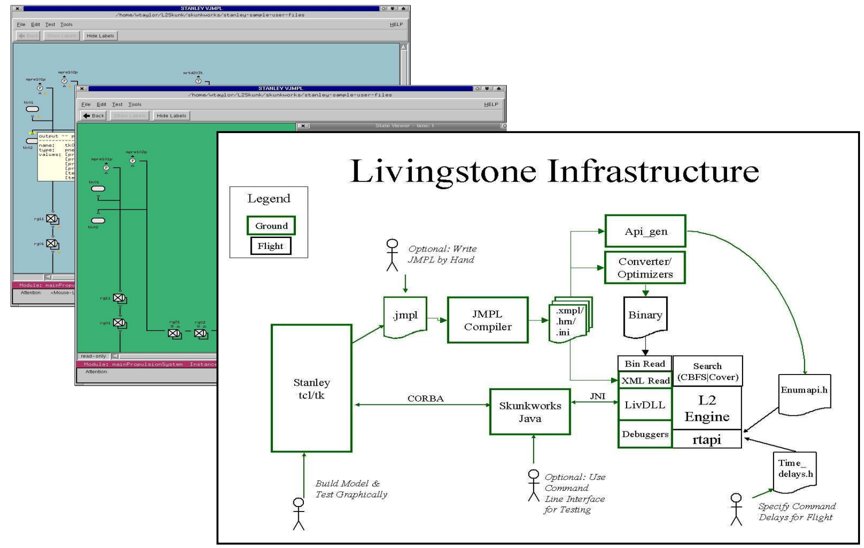
Mark Shirley/ARC

Goal: Create modeling and debugging tools for model-based programming of autonomous systems

Work focused on monitoring, diagnosis & recovery portion of agent

Key Deliverables:

- Visual modeling language, more engineer-friendly textual syntax
- Application of formal V&V techniques to model-based autonomy
- Rapid prototyping of scenarios



NASA Relevance:

- Facilitate transition of model-based programming into a sustainable engineering practice
- Reduce flight software development costs; increase flight software robustness

Customers:

- X-34 Experimental RLV (NITEX/PITEX flight experiment)
- X-37 Experimental RLV
- Honeywell and Interface Control Systems RLV2 team
- Northrup/Grumman RLV2 team

Schedule:

- Project ending in FY02

Proposed next steps:

- Modeling and Debugging tools for Planning
- Simulation-based fault insertion testbed for K9 arm
- Model-checking work picked up by another R&D program

backups

Relationship to Mission Sim Facility

- Candidate for physics simulation
- Made sure it's compatible with Viz
- Not just for rovers; PSA, etc

Livingstone Progress Summary

Monitoring (fault detection)

- ✓ – Discrete dynamics
- ☆ – Diagnostic cycle management (timeouts, overlapping commands)
- – Hybrid dynamics
- – Performance parameter estimation

Fault diagnosis

- ✓ – Single hypothesis interface to rest of agent
- ☆ – Multiple hypotheses interface to rest of agent
- ☆ – Long-lead time diagnoses
- – Information-gathering actions

Command sequence generation

- – Safing
- ✓ – Recoveries

Interaction with the ground

- ☆ – Limited visibility of commands onboard
- ☆ – Limited downlink bandwidth

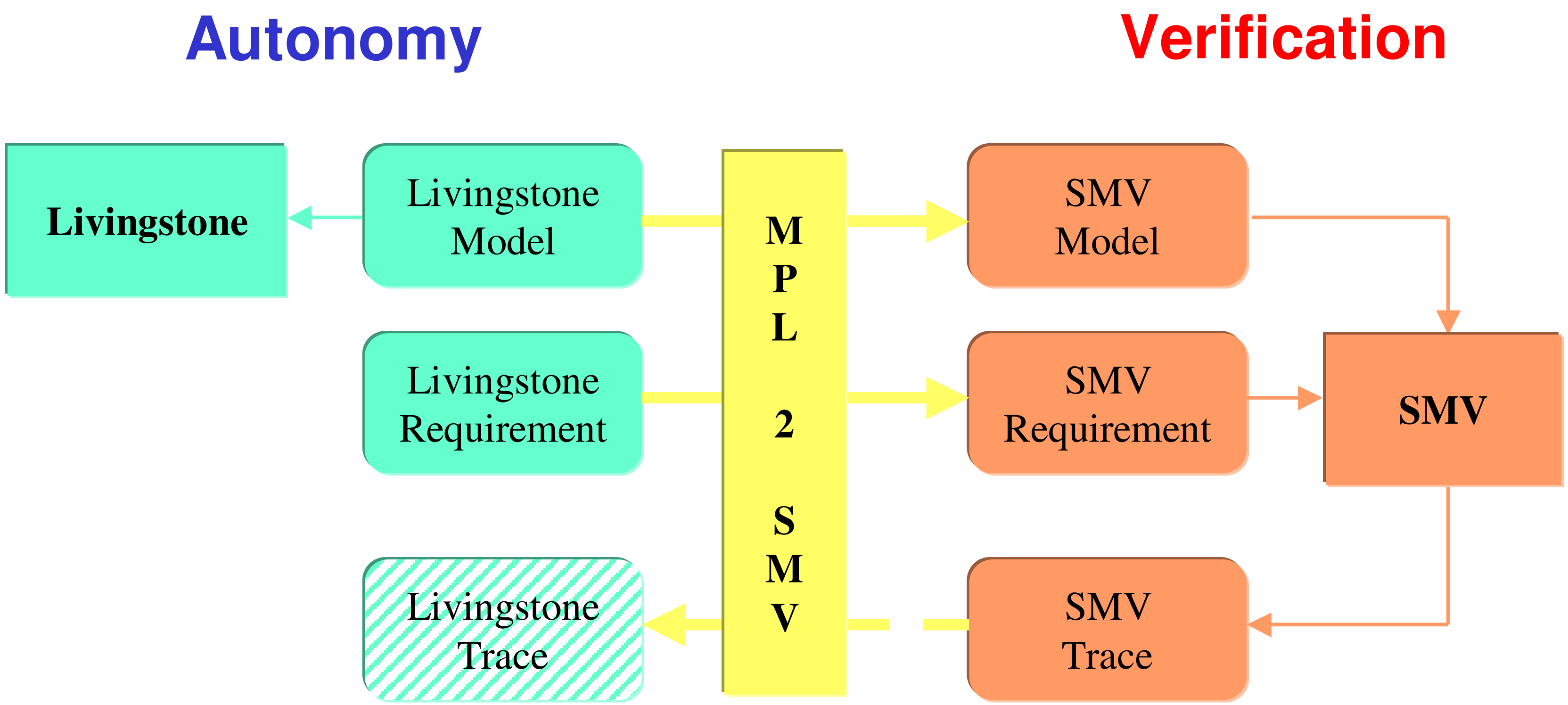
Software engineering

- ☆ – Integration with flight control software
- – Process executed by a non-experimental design team

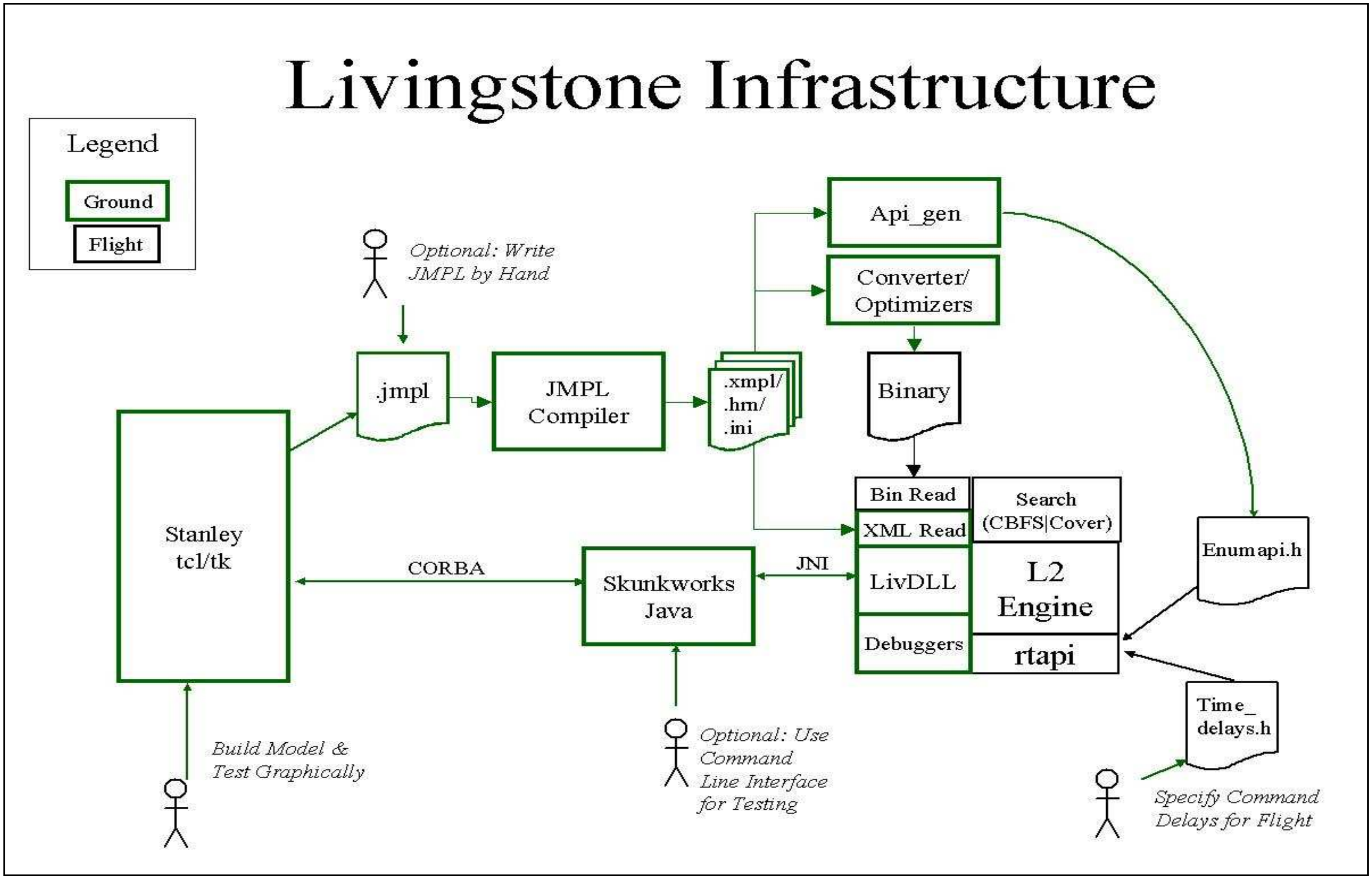
✓ Demonstrated by RAX

☆ Progress since RAX

MPL2SMV


Autonomy
Verification
Livingstone
Livingstone Model
Livingstone Requirement
Livingstone Trace
MPL 2 SMV
SMV Model
SMV Requirement
SMV Trace
SMV

Livingstone (L2) + Skunkworks Flow Chart



FDIR for the International Space Station (ISS) using Model-based Reasoning (L2)

OBJECTIVES

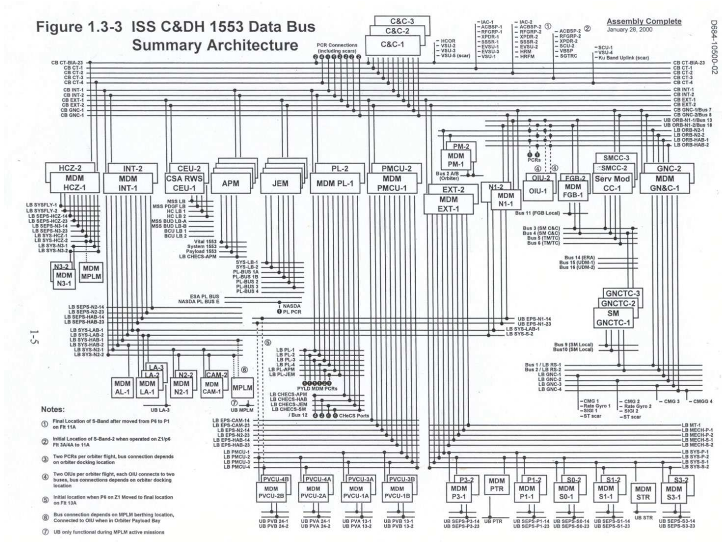
- To develop model-based reasoning technology for FDIR of the Command and Data Handling (C&DH) subsystem of ISS.

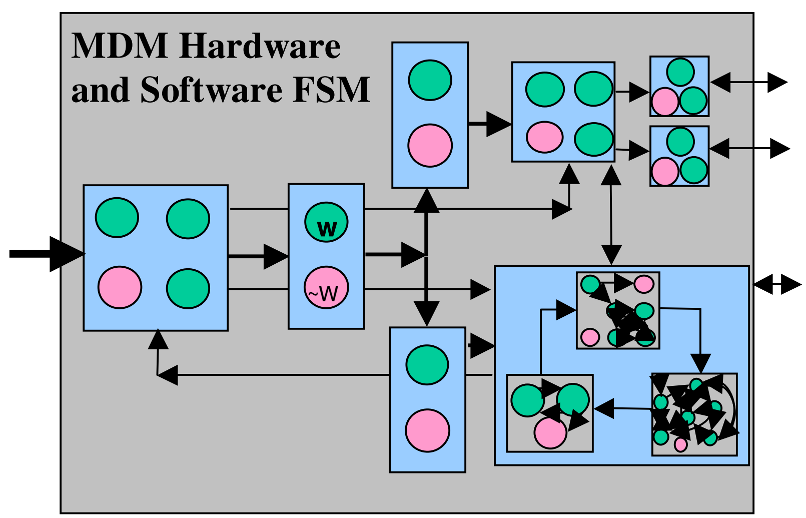
BENEFITS

- Increase ISS safety and science at a time when ISS budgets are decreasing and loads on ISS C&DH are still increasing.
- Provide foundation for IVHM – all subsystems use C&DH for sense/act – SLI will leverage ISS.
- Determine utility of using model-based reasoning to model software processes in conjunction with hardware.

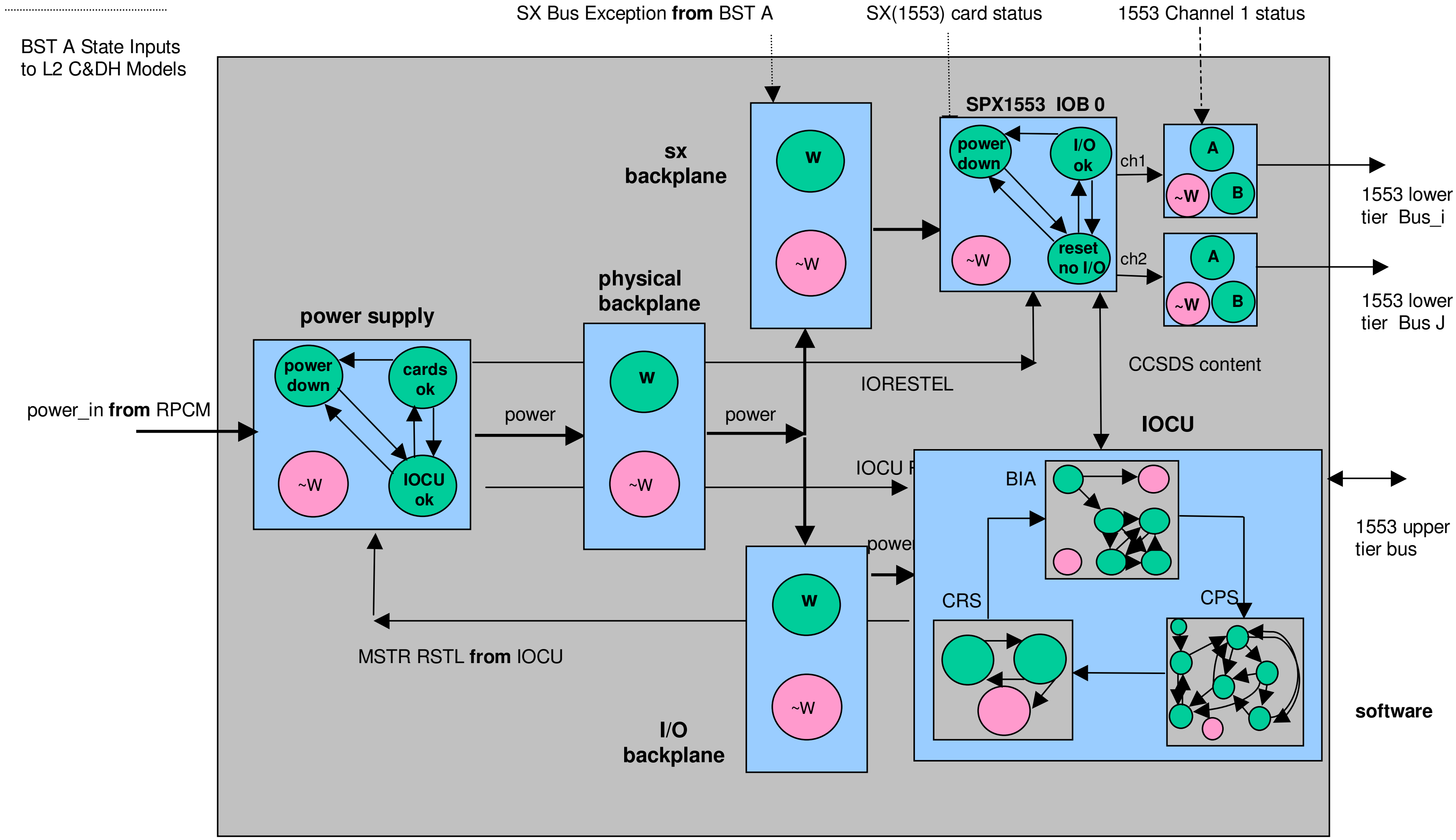
APPROACH

- Three phases: 1) offline analysis of ISS data dumps, 2) realtime ground ops, 3) realtime ISS ops.
- Leverage ISS Caution and Warning (C&W) system as monitors to L2 models.
- Model hardware: computers and buses.
- Model software: 1) memory locations as containers, 2) software functions as components whose ports are inputs/outputs of software, 3) qualitative rate monotonic scheduler.





MDM Module



MDM module is made up of collection components including PS, SX Backplane, I/O backplane, I/O Cards, SPD1553 Cards, IOCU card.